04

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/693,633	10/23/2003	Joseph S. Beda	3481	8889

7590 08/08/2006

Law Offices of Albert S. Michalik, PLLC
Suite 193
704 -228th Avenue NE
Sammamish, WA 98074

EXAMINER	
WOODS, ERIC V	
ART UNIT	**PAPER NUMBER**
2628	

DATE MAILED: 08/08/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No. 10/693,633	Applicant(s) BEDA ET AL.	
	Examiner Eric Woods	Art Unit 2628	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on 27 April 2006.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) 1-35 is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) 1-35 is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on ____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

Election/Restrictions

Applicant's election without traverse of claims 1-35 (Group I) in the reply filed on 4/27/2006 is acknowledged. While the election was not specified to be with or without traverse, applicant's cancellation of claims 36-62 and lack of any statements traversing the requirement result in this being treated as an election without traverse as per Office policy on the matter.

Response to Arguments

Applicant's arguments, see Remarks pages 1-11 and page 1, filed on 9/27/2005 and 11/30/2005 respectively, with respect to various grounds of rejection have been fully considered and are persuasive.

All rejections of claims 36-62 stand withdrawn since applicant canceled those claims.

All claims are given effective date of the filing date of the instant application, as per reasons stated in previous Actions.

The rejections of claims 1-35 under 35 USC 103(a) stand withdrawn in view of applicant's amendments.

However, upon further consideration, new grounds of rejection follow below.

Applicant is recommended to amend the independent claims to include at least the limitations that the graphics are two-dimensional and vector in nature. Examiner would also respectfully suggest adding some more detail to the independent claims in terms of how the API relates the scene graph (i.e. direct access vs. indirect access).

If applicant were to make such amendments, they probably would result in the withdrawal of the current grounds of rejection against the instant claims. Applicant is encouraged to call examiner to discuss this.

Claim Objections

Claim 1 is objected to because of the following informalities:

Line 4 – the recitation "a markup language data" is not correct. The term 'data' is plural – the correct singular form is 'datum,' unless the intent was for a plurality, wherein the 'a' would be incorrect. Appropriate correction is required.

Claim Rejections - 35 USC § 112

The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

Claims 4-5, 17-19, and 27-29 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. See below for specific listing of claims

Specifically, claims 4-5, 17-19, and 27-29 are rejected because the term "a visual" is used where it is unclear what the terminology is referring to and the specification does not reasonably apprise one of ordinary skill in the art what the definition would be. Note that in other claims, 'visual objects,' 'visual classes,' and the like are referred to, so it is truly unclear and indefinite. **For examination purposes, examiner will treat "a visual" as meaning "a visual object or element" in the context of a

visual element in SVG specification, which applicant clearly states is being used in his invention.

Claim 15 is rejected because the term 'path' is used, and it is unclear what the meaning of 'path' – whether applicant is referring to the path of a visual element during an animation or a path as defined in the SVG specification.

Claims 5, 26, 27, and 59 are rejected because the term 'context' is used, and it is unclear what the meaning of 'context' is in this claim. In the SVG specification, the term 'context' is not specifically set forth and applicant does not provide a clear basis of the definition. Context can relate to the device-specific drawing information and capabilities, as set forth by various references; it can be a term relating specifically to the properties and metadata associated with various visual elements; it can be many things. As such, particularly with respect to the graphics art, it is critically important to be able to determine the implication of the word, because the 'context' of a device for rendering purposes is very different that the 'context' of a data node in a scene graph, etc.

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

The factual inquiries set forth in *Graham* **v.** *John Deere Co.*, 383 U.S. 1, 148 USPQ 459 (1966), that are applied for establishing a background for determining obviousness under 35 U.S.C. 103(a) are summarized as follows:

1. Determining the scope and contents of the prior art.
2. Ascertaining the differences between the prior art and the claims at issue.
3. Resolving the level of ordinary skill in the pertinent art.
4. Considering objective evidence present in the application indicating obviousness or nonobviousness.

Claims 1-4, 18, 20, and 23-24 are rejected under 35 U.S.C. 103(a) as being unpatentable over Itoh in view of Walczak (Walczak, K. and W. Cellary - "Building Database Applications of Virtual Reality with X-VRML") and Eleftheriadis (US 6,092,107 A1).

As to claim 1,

In a computing environment, a computer-implemented method comprising:

-Receiving a function call via an application programming interface, the function call comprising a native format including a markup language data; (Walczak teaches an XML-based programming language called XVRML (see page 112, third paragraph on the left, and section 2.1 on the right). This markup is interpreted (see section 2.1)(see page 116, section 3.1, teaches an implementation of X-VRML; the VRML interpreter takes the input file, analyzes it, and produces the final code. The X-VRML file can **contain the whole model of the virtual scene** (e.g. an element tree)(section 3.1, page 116), where X-VRML file contains hierarchical data, because it contains VRML, which is known to contain hierarchies, which therefore constitutes an element tree, and clearly a X-VRML model is an element object model, especially given that various portions of an

entire object model can exist in different places (e.g. X-VRML representations of only certain objects within a scene, such as for a web-based store as posited in section 3.1 on page 116). Clearly such code is in native format. Next, VRML code contains property data for objects, see as an example Figure 1, where specific items – such as shapes – are provided with specific characteristics and the like.) (Eleftheriadis (3:1-30) emphasizes: "The use of application programming interfaces (APIs) has been long recognized in the software industry as a means to achieve standardized operations and functions over a number of different types of computer platforms... In the field of graphics, Virtual Reality Modeling Language (VRML) allows a means of specifying spatial and temporal relationships between objects and description of a scene by use of a scene graph approach.... To enhance features of VRML and to allow programmatic control, DimensionX has released a set of APIs known as Liquid Reality. Recently, Sun Microsystems has announced an early version of Java3D, an API specification which among other things supports representation of synthetic audiovisual objects as scene graphs...")

-Interpreting the markup language data in its native format to cause data in a scene graph to be modified; and (Walczak - VRML inherently utilizes scene graphs; indeed, as stated above, VRML utilizes a scene graph programming model, and maintains the scene graph internally. The VRML scene instance in section 3 on the right side in Figures 3-8 constitutes a scene graph-containing instance. **Note that the system can take base files of VRML, X3D, or other well-known file formats**, where X3D is the successor format to VRML – section 2.1, page 112.)(Walczak – as recited above,

clearly the VRML data is interpreted in its native format, as described above, concerning the nature of the X-VRML language, how it is XML but still native format language. Clearly, any changes it causes to the scene graph. Walczak clearly translates some of the elements and property data in the element tree into requests to the scene graph interface layer. Specifically, in section 2.1, Walczak specifies that the code located inside a repeating element is interpreted multiple times, which would cause multiple elements to be added to the scene graph, see as an example the sample code provided in Figure 1, where the interpreted code repeatedly sent requests to add additional cylinders to the scene graph, with the resultant output shown in Figure 2, where clearly this shows that multiple requests are sent to the scene graph to add multiple objects)

-Causing a change in a display in response to the modification of data in the scene graph. (Walczak contains a display device – see section 3.1, where there is discussion of content being displayed, and in Figures 9-11 on page 118, because the various graphics are rendered and shown on a screen to the viewer, there must be such information)

Walczak teaches all the limitations of the above claims, except for explicitly teaching that VRML uses a scene graph (which is inherent to VRML in any case, and Eleftheriadis teaches that it does 3:1-30) and for suggesting that the scene graph layer utilize an API. The Eleftheriadis reference teaches that the use of APIs for scene graphs is well known in the art and that it is the solution that the prior art had come to with regards to enhancing functionality. Finally, Eleftheriadis itself utilizes plural APIs to access the scene graph, amongst other things (see 4:35-50), where the Scene Graph

layer explicitly has its own API (5:37-60). The two references are analogous arts. Next, clearly the X-VRML language is similar to that of the Java language (see Table 1, page 112) in syntax, so the use of similar programmatic structures for the same problem would be extremely obvious, not to mention that X-VRML is implemented in Java (page 117). Therefore, in light of the teachings of Eleftheriadis, it would have been obvious to one of ordinary skill in the art at the time the invention was made to modify Walczak to utilize an API (possibly that of Eleftheriadis) for the scene graph for at least the reasons already set forth.

As to claim 2, clearly adding objects to the scene graph causes the creation of a new visual class, where X-VRML clearly teaches the existence of a class-based language object-oriented language (Walczak, all of page 112). Each object is its own class, as shown in Figure 1, and the example in Figure 2, where the X-VRML graph generates multiple instances of the shown items and initializes them with certain properties as specified in the code in Figure 1 on page 115.

As to claim 3, clearly Walczak teaches the use of transforms with a visual object in the graph using X-VRML and VRML – see Figure 1, with the Transform object applied to the cylinders as noted in the inline documentation and the like, which is an association with a visual object, and clearly the code in Figure 1 invokes such transforms.

As to claim 4, Walczak clearly invokes code in Figure 1 that causes elements to be added to the scene graph that must prima facie exist (VRML 2.0 / VRML97 (utilized by the Walczak system) inherently has a scene graph with a base node, and

Eleftheriadis also teaches the scene graph with the API)), where clearly the code in Figure 1 places new instances of shapes into the scene graph representing the scene in Figure 2 and the like.

As to claim 18, clearly Walczak shows such transforms in the code shown in Figure 1.

As to claim 20, clearly the teachings of Eleftheriadis concerning the scene graph API would lead one of ordinary skill in the art to understand that there would be a common API to and for elements in the scene graph, and this is also the case in VRML (Examiner takes Official Notice of the fact that VRML utilizes a scene graph, and the Eleftheriadis presents the direct idea of having the easily accessible scene graph).

As to claims 23 and 24, clearly VRML teaches image resources such as image textures as in 6.22 and 6.33. It is further clear from the VRML specification that such image textures are applied to shapes, which clearly meets the requirement that such an image is associated with an image effect object, which in VRML it would be.

As to claim 25, VRML teaches a text node in section 6.47.

As to claim 28, VRML teaches 3D nodes, see as an example Walczak Figures 1 and 2 and the cylinder objects, which are three-dimensional (see VRML section 6.14 as an example).

As to claim 29, VRML teaches mapping image textures onto objects and the like – see 6.22, 6.33, 6.48, and 6.49, where this clearly constitutes mapping a two-dimensional visual onto a three-dimensional surface.

As to claim 30, VRML teaches Interpolator Nodes in 4.6.8 and Time-Dependent Nodes in 4.6.9 where these are designed for linear keyframed animation, and all the sensor objects (6.49-6.54) are used for animation, and this is explicitly discussed in the VRML specification, and so constitutes 'invoking a function' to insert animation data. Walczak also discusses animation and the like in terms of VRML objects.

Claims 5, 26-27l, and 32-35 are rejected under 35 USC 103(a) as unpatentable over Walczak (which teaches VRML) and Eleftheriadis as applied to claim 1, and further in view of Itoh and the SVG standard (which Itoh teaches) and Steele.

As to claim 5,

The method of claim 4 further comprising, causing drawing context to be returned, the drawing context providing a mechanism for rendering into the drawing visual.

References Walczak and Eleftheriadis do not expressly teach this limitation, but clearly teaches that users navigate through a virtual environment, which *prima facie* requires that device specific information regarding display information, e.g. drawing context, be available to the rendering engine. Reference Steele teaches this limitation, as for example he teaches the retrieval of device context in [0101]. Clearly, the device receives information based on its device context, which clearly is associated with the drawing context, as the two are one and the same in this case. Steele teaches rendering in [0007 and 0011-0012]. The drawing context per se is incorporated into the data structures of Steele (see Figure 7). Reference Walczak and Eleftheriadis clearly

teach rendering (Walczak, Figures 1 and 2, and the like). Further, Itou teaches the use of scene graph data hierarchies (see Fig. 21), which clearly establishes that drawing context must exist in order for that group node to know its spatial location (29:60-30:25). It further would have been obvious to one of ordinary skill in the art at the time the invention was made to modify the system of Walczak / Eleftheriadis to utilize a device specific context so as to optimize data streamed to that device for purposes of minimizing memory consumption (a large problem pointed out by Steele [0007]).). (Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task.

However, VRML does teach the use of billboard objects (section 6.6), image textures (6.22, 6.33) and the like, which involve two-dimensional graphics. Clearly, use of such two-dimensional graphics as source information would be important, particularly where such data comes in a standard format, such as SVG. Itoh teaches a browser mechanism that can be implemented using Java and receive input files from different file formats, such as VRML and SVG, and combine them to create a three-dimensional world with such elements. It is preferable to do so because all functions are written in a descriptive language (such as Java or the X-VRML variant) that allows integrating two- and three-dimensional drawing primitives and the like (see 1:20-3:30). Next, it is preferable to mix such documents as specified therein for the various advantages, where a unified environment is beneficial, and VRML and X-VRML do not expressly provide methods for incorporating two-dimensional drawing commands and the like. It would have been obvious to one of ordinary skill in the art at the time the invention was

made to modify Walczak/Eleftheriadis/VRML to allow the handling of two-dimensional drawing content for the above reasons, and for the additional flexibility afforded by incorporating such additional files into the three-dimensional environments as recited by Itoh.

Therefore, such a combination would handle SVG graphics, and drawing commands from SVG would therefore be relevant.

As to claim 26,

Clearly Steele teaches providing a context for rendering such that it can more effectively rendered for the device or platform that it is on. Motivation and combination are taken from the rejection to claim 5 above. Reference Steele teaches this limitation, as for example he teaches the retrieval of device context in [0101]. Clearly, the device receives information based on its device context, which clearly is associated with the drawing context, as the two are one and the same in this case. For the second case, if the definition of context is the data associated with a specific element – e.g. the details of the element, its location, color, et cetera, these attributes are inherent in SVG elements as set forth in the rejections above, e.g. sections 11.1, 9.1-9.7, et cetera. Further, Steele teaches the same in Figure 7, where each element has certain properties that would be a drawing context, in the sense that each visual element has those properties associated with it [Steele 0052-0056 and 0059-0061]. Steele clearly teaches data modification in [0061] as set forth above. Itou does not expressly teach this limitation.

As to claim 27,

The method of claim 26 wherein the function call corresponds to a retained visual, and further comprising, calling back to have the drawing context of the retained visual returned to the scene graph data structure.

References Itou, Walczak, Eleftheriadis, and Steele do not expressly teach these limitations; references Steele and Itou teach them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification clearly sets forth that all elements (as well as 3.1 and 4.2) have properties associated with them. The system of Steele clearly caches visuals during processing – see [0083], and it would be obvious that such data would be pulled from the cache to find out the state and properties of a visual element. Steele clearly teaches data modification in [0061] as set forth above. Further, it would be obvious to one of ordinary skill to so modify the Kim reference to cache the visuals so that they would be retained and that data would be pulled from the cache as set forth above, and as the Steele reference sets forth to have it pulled from there during data processing, including that of data trees like unto the one in Figure 7, as in [0100 Steele]. As such, references Itou and Steele intrinsically teach this limitation. Motivation for combining the references is taken from the rejections to claims 5 and 26 above.

Steele obviously renders the visuals to be displayed to the user. Steele clearly establishes in [0051-0054] and Figs. 6 and 9 that animation takes place through the SVG standard. Section 19.2 of SVG sets forth how this takes place, and at the bottom three paragraphs of section 19.2.2 it clearly states that animation has a document start and document end, and further in the second to last paragraph that the SVG system

indicates the timeline position of document fragments being animated. Further, according to SVG 19.2.2 the animation is by document fragment and object path, which clearly are passed to the system is specified in, for example, the second page of Fig. 9 in the SVG XML program. Clearly, the system of Steele performs compositing and rendering [0007, 0011-0012]. Finally, reference SVG teaches that it supports compositing (section 14.2.1). The composition engine would be that of Steele. Motivation the combination is taken from the rejection to claim 5 above.

As to claim 32,

The method of claim 31 wherein the composition engine interpolates graphics data based on the timeline to animate an output corresponding to an object in the scene graph data structure.

References Walczak, Itou, and Eleftheriadis do not expressly teach this limitation, while reference Steele does teach it. Steele in Figs. 6 and 9 shows animated elements [0041] and in Fig. 8, clearly during an animation the actions are shown, where the system in steps 835, 840, and 845 performs interpolation for nodes shown in the tree in Fig. 7. Clearly interpolation takes place during animation [0072, 0077-0079] which performs interpolation during the animation process as set forth in the SVG standard, and *prima facie* the output would only be objects in the scene graph, and they would *prima facie* be based on the timeline as set forth in the rejection to claim 31 above (VRML provides timelines, note time-dependent nodes in 4.6.9 and the like). Steele also teaches more detailed animation coding. Motivation for combination is taken from the rejection of claim 5 above.

As to claim 33,

The method of claim 32 wherein the composition engine is at a low-level with respect to the scene graph.

References Walczak, Eleftheriadis, and Itou do not expressly teach this limitation, while reference Steele does teach it. Steele in Fig. 15 shows that various programs, including the operating system, are on the flash memory, which in [0136] is specified to contain all the low-level programs of the operating system – graphics is low-level functionality. Since there is no specific graphics unit, all graphics operations and compositing are done by the operating system in the microprocessor, which *prima facie* means that in that embodiment, such graphics are done at a low-level, that is the rendering is done by the operating system at a low level. The scene graph is high-level in that it is embodied in RAM and is held as an abstraction – this is a function of the SVG standard that keeps tree nodes and container nodes as abstract as possible, therefore the embodiment in Fig. 18 must do the same. In any case, low-level composition means that it would be done by hardware that is much faster than software. As such, it would be obvious to modify the device of Steele to use low-level rendering, and in any case Steele has the rendering means set forth in the rejection to claim 32 above, which is *prima facie* entirely hardware. Motivation is taken from the rejections to claims 4 and 5.

As to claim 34,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place an object corresponding to audio and/or video data into the scene graph data structure.

References Itou and Steele do teach it. Itou teaches this limitation in Fig. 5. Steele in Figs. 8 shows audio elements 820 and 830 in the animation execution and in Fig. 7 a scene graph data structure (a tree). Steele [0050, 0052] for example provides that such animation takes place, and the SVG standard in 6.18 clearly sets forth aural style sheets, that are audio data that each element can have animation associated with it, which clearly is placed into the scene graph of Fig. 7. Also, by definition, SVG animations would be video. Motivation for combining is taken from the rejection to claim 5 above.

As to claim 35,

The method of claim 1 wherein causing data in the scene graph to be modified comprises changing a mutable value of an object in the scene graph data structure.

Steele teaches in [0014] that one embodiment of his invention changes the visual graph in accordance to changes of the sequence graph, where the visual graph is comparable to the "scene graph" of applicant and mutable values (e.g. position) of elements in the tree are shifted as per Steele [0052-0057]. Therefore, the limitation is met, and it would have been obvious to modify it so that it in fact change mutable values on the tree if applicant feels that this is not an adequate embodiment of this particular limitation. Motivation for the combination is taken from the rejections to claims 4 and 5.

Claims 6-22 are rejected under 35 USC 103(a) as unpatentable over Walczak and Eleftheriadis as applied to claim 1 above, and further in view of Itoh and SVG.

As to claim 6,

The method of claim 2 wherein causing data in the scene graph to be modified comprises invoking code to associate brush data with a visual object in the scene graph.

Reference Walczak and Eleftheriadis do not expressly teach this limitation. Reference Itou does teach this limitation by the use of SVG graphics. Turning to the SVG (, section 11 titled 'Painting: Filling, Stroking, and Marker Symbols', specifically section 11.1, 'With SVG, you can paint (e.g. stroke or fill) with: ..." and then proceeds to list several. The term 'brush data' is clearly analogous to the 'paint' operation in SVG with comparable data. Given that SVG allows (from section 11.1) a single color, a solid color (with or without opacity), a gradient, a pattern (vector or image), and custom patterns, clearly each visible element clearly has such data associated with it (see section 11.2 in its entirety, 11.7 for specific properties, section 11.8 for how painting properties can be inherited, which *prima facie* justifies the position that element have intrinsic painting properties, i.e. brush data as set forth above. Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task, and SVG data element *prima facie* and inherently

possess paint data as set forth by the SVG specification above. As such, references Itou and Steele intrinsically teach this limitation and the SVG standard inherently handles these paint limitations. Motivation and combination is further taken from claim 5 and (relevant portions) incorporated by reference.

However, VRML does teach the use of billboard objects (section 6.6), image textures (6.22, 6.33) and the like, which involve two-dimensional graphics. Clearly, use of such two-dimensional graphics as source information would be important, particularly where such data comes in a standard format, such as SVG. Itoh teaches a browser mechanism that can be implemented using Java and receive input files from different file formats, such as VRML and SVG, and combine them to create a three-dimensional world with such elements. It is preferable to do so because all functions are written in a descriptive language (such as Java or the X-VRML variant) that allows integrating two- and three-dimensional drawing primitives and the like (see 1:20-3:30). Next, it is preferable to mix such documents as specified therein for the various advantages, where a unified environment is beneficial, and VRML and X-VRML do not expressly provide methods for incorporating two-dimensional drawing commands and the like. It would have been obvious to one of ordinary skill in the art at the time the invention was made to modify Walczak/Eleftheriadis/VRML to allow the handling of two-dimensional drawing content for the above reasons, and for the additional flexibility afforded by incorporating such additional files into the three-dimensional environments as recited by Itoh.

Therefore, such a combination would handle SVG graphics, and drawing commands from SVG would therefore be relevant.

As to claim 7,

The method of claim 6 wherein the brush data comprises receiving data corresponding to a solid color.

References Itou, Walczak and Eleftheriadis do not expressly teach these limitations; references Itou teach them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with a solid color with opacity, thus meeting this limitation. Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task, and SVG data element *prima facie* and inherently possess paint data as set forth by the SVG specification above. As such, references Itou and Steele intrinsically teach this limitation. Motivation and combination are further taken from claim 6 above and incorporated by reference.

As to claim 8,

The method of claim 6 wherein receiving brush data comprises receiving data corresponding to a linear gradient brush and a stop collection comprising at least one stop.

References Itou, Walczak and Eleftheriadis do not expressly teach these limitations; reference Itoh teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets

forth that a user can paint with a gradient that can be linear. Further, sections 11.7.1 and 11.7.2 of the specification sets forth that gradient stops are included in the SVG 'color-interpolation' property. As such, reference Itou intrinsically teaches this limitation; motivation and combination is taken from 6 and incorporated via reference.

As to claim 9,

The method of claim 6 wherein receiving brush data comprises receiving data corresponding a radial gradient brush.

References Itou, Walczak and Eleftheriadis do not expressly teach these limitations; reference Itou teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with a gradient that can be radial and also see sections 11.7.1 and 11.7.2 for more detail, thus meeting this limitation. As such, reference Itou intrinsically teaches this limitation; motivation and combination is incorporated from claim 6.

As to claim 10,

The method of claim 6 wherein receiving brush data comprises receiving data corresponding to an image.

References Itou, Walczak and Eleftheriadis do not expressly teach these limitations; reference Itou teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with an image with further details provided in section 11.7.5

under the 'image-rendering' property. Motivation and combination is incorporated from claim 6.

As to claim 11,

The method of claim 10 further comprising, receiving markup corresponding to an image effect to apply to the image.

References Itou, Walczak and Eleftheriadis do not expressly teach these limitations; reference Itou teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 14.4 of the SVG specification sets forth that a user can use any image as an opacity mask, thus meeting this limitation, given that alpha blending is *prima facie* an image effect. Further, note that the rendering portion of Itou would receive such information (the rendering functionality is inherent in SVG – see section 11.7, 14.4, et cetera). Motivation and combination is also incorporated from claim 6.

As to claim 12,

The method of claim 1 further comprising, receiving markup corresponding to pen data that defines an outline of a shape.

References Itou, Walczak and Eleftheriadis do not expressly teach these limitations; reference Itou teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. The term 'pen data' used by applicant above is comparable or analogous to any set of data defining the outline of a shape, including SVG 'path' data. Section 11.3 of the SVG specification sets forth that a user can fill a path that would correspond to the outline of shape with multiple illustrations

provided for this under the 'nonzero' and 'even odd' subheadings – see details on paths -- with further details provided in section 11.3 and 11.4 (the individual strokes that create these effects. SVG is a markup language, therefore any SVG rendering utility would *prima facie* receive markup. Motivation for combining Itou, Walczak and Eleftheriadis, and SVG is taken from the rejection to claim 6.

Also, VRML teaches a shape node, where these are found in section 4.6.3, and 6.41, where a shape node contains an 'appearance' node and a 'geometry' node, where the geometry node is of type defined in 4.6.3, which are clearly shapes.

As to claim 13,

The method of claim 1 wherein the markup corresponds to a shape class comprising at least one of the set containing rectangle, polyline, polygon, path, line and ellipse shapes.

References Itou, Walczak and Eleftheriadis do not expressly teach these limitations; reference Itou teach them intrinsically as set forth above in claim 1 and reference SVG clearly supports this position. The SVG specification sets forth classes of shapes in section 9.1, where all six of the recited shapes (rectangle, polygon, path, line, polyline, and ellipse) are set forth. Motivation for combining Itou, Walczak and Eleftheriadis, and SVG is taken from the rejection to claim 6.

As to claim 14, Walczak/Eleftheriadis/Itoh do not expressly teach this limitation. It corresponds to the 'rect' element in the SVG specification 9.2 as set forth above in the rejection to claim 1. Motivation is taken from the above.

As to claim 15, Walczak/Eleftheriadis/Itoh do not expressly teach this limitation, but Itoh implicitly teaches it by use of the SVG language, wherein SVG teaches the use of path elements in SVG section 9.1, where it is stated that basic shapes are equivalent to a 'path' element that would construct them; path elements are taught in detail in sections 8.1-8.5, specifically 8.1 and 8.3.1 for example. Motivation and combination is taken from claim 6 and herein incorporated by reference.

As to claim 16, Walczak/Eleftheriadis/Itoh do not expressly teach this limitation, but Itoh implicitly teaches it by use of the SVG language, wherein SVG teaches the use of line elements in SVG section 9.5. Motivation and combination is taken from claim 6 and herein incorporated by reference.

As to claim 17, Itou teaches SVG, and the SVG specification sets forth hit testing in section 16.6 (the two paragraphs right at the end of the section) where hit testing (namely, hit detection) is taught, specifically testing text for character or cell hits and testing visual elements for hits in their entirety, and such information is clearly communicated in markup language – see section 16.2 for event types and elements transmitted in markup, for example.

As to claim 18, clearly, Itou teaches in 29:60-30:25 that transform nodes are applied to transforming spatial coordinates of objects in the scene graph in the form of child nodes and the rest of it.

As to claim 19, references Walczak, Eleftheriadis, and Itou do not expressly teach these limitations; reference Itou teach them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Bounding box calculations are

taught in section 7.1 and detailed in section 7.11 where they are calculated. Walczak clearly teaches data modification in Figures 2-8 as set forth above. In Itou, clearly the scene graph can include line objects since it can include SVG. As such, reference Itou intrinsically teaches this limitation. Motivation for combining Itou, Eleftheriadis, and Walczak is taken from the rejection to claim 6.

As to claim 20,

The method of claim 1 wherein causing data in the scene graph be modified comprises invoking a function via a common interface to a visual object in the scene graph data structure.

Clearly, such modifications to the scene graph are made via the Java API and the scene graph API itself to allow the objects to be modified as set forth in the rejection to claim 6 above. As such, the rejection to claim 6 is incorporated by reference.

As to claim 21, further, Itou teaches in Fig. 21 the use of trees as set forth there, and teaches manager objects in 2:50-67. SVG teaches that all implementations must implement a rendering model as set forth in 3.1 and so forth, and scene graphs are known to directed acyclic, i.e. trees. Clearly, this model is implemented through the DOM interfaces set forth in section 4, and each element has its own element information that controls rendering aspects.

As to claim 22, references Walczak, Eleftheriadis, and Itou do not expressly teach these limitations; reference Itou teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. SVG clearly teaches the use of container objects, as in section 1.6 it clearly teaches the use of 'container elements',

which are defined as 'An element that can have graphic elements and other container elements as child elements'. Walczak clearly teaches data modification on pages 112-115 as set forth above. Clearly, the container object could be the head object of the tree structure shown in and as the group node of Itou, shown as the head node in Fig. 21. As such, reference Itou intrinsically teaches this limitation. Motivation for combining Itou, Kim, and SVG is taken from the rejection to claim 6.

Claims 1, 4-17, 19, 21-27, and 30-35 are rejected under 35 U.S.C. 103(a) as being unpatentable over Steele/SVG specification in view of Walczak (Walczak, K. and W. Cellary - "Building Database Applications of Virtual Reality with X-VRML"), Eleftheriadis (US 6,092,107 A1).

As to claim 1,

In a computing environment, a computer-implemented method comprising:

-Receiving a function call via an application programming interface, the function call comprising a native format including a markup language data; (Walczak teaches an XML-based programming language called XVRML (see page 112, third paragraph on the left, and section 2.1 on the right). This markup is interpreted (see section 2.1)(see page 116, section 3.1, teaches an implementation of X-VRML; the VRML interpreter takes the input file, analyzes it, and produces the final code. The X-VRML file can **contain the whole model of the virtual scene** (e.g. an element tree)(section 3.1, page 116), where X-VRML file contains hierarchical data, because it contains VRML, which is known to contain hierarchies, which therefore constitutes an element tree, and clearly a

X-VRML model is an element object model, especially given that various portions of an entire object model can exist in different places (e.g. X-VRML representations of only certain objects within a scene, such as for a web-based store as posited in section 3.1 on page 116). Clearly such code is in native format. Next, VRML code contains property data for objects, see as an example Figure 1, where specific items – such as shapes – are provided with specific characteristics and the like.) (Eleftheriadis (3:1-30) emphasizes: "The use of application programming interfaces (APIs) has been long recognized in the software industry as a means to achieve standardized operations and functions over a number of different types of computer platforms... In the field of graphics, Virtual Reality Modeling Language (VRML) allows a means of specifying spatial and temporal relationships between objects and description of a scene by use of a scene graph approach.... To enhance features of VRML and to allow programmatic control, DimensionX has released a set of APIs known as Liquid Reality. Recently, Sun Microsystems has announced an early version of Java3D, an API specification which among other things supports representation of synthetic audiovisual objects as scene graphs...")(Steele clearly teaches that commands are received in markup language – that is what SVG is – see Figure 9 for examples of the code)

-Interpreting the markup language data in its native format to cause data in a scene graph to be modified; and (Walczak - VRML inherently utilizes scene graphs; indeed, as stated above, VRML utilizes a scene graph programming model, and maintains the scene graph internally. The VRML scene instance in section 3 on the right side in Figures 3-8 constitutes a scene graph-containing instance. **Note that the system can**

take base files of VRML, X3D, or other well-known file formats, where X3D is the successor format to VRML – section 2.1, page 112.)(Walczak – as recited above, clearly the VRML data is interpreted in its native format, as described above, concerning the nature of the X-VRML language, how it is XML but still native format language. Clearly, any changes it causes to the scene graph. Walczak clearly translates some of the elements and property data in the element tree into requests to the scene graph interface layer. Specifically, in section 2.1, Walczak specifies that the code located inside a repeating element is interpreted multiple times, which would cause multiple elements to be added to the scene graph, see as an example the sample code provided in Figure 1, where the interpreted code repeatedly sent requests to add additional cylinders to the scene graph, with the resultant output shown in Figure 2, where clearly this shows that multiple requests are sent to the scene graph to add multiple objects)

-Causing a change in a display in response to the modification of data in the scene graph. (Walczak contains a display device – see section 3.1, where there is discussion of content being displayed, and in Figures 9-11 on page 118, because the various graphics are rendered and shown on a screen to the viewer, there must be such information)(Steele provides changes in the scene graph based on the SVG data – the SVG data is interpreted – see Figure 3, where the SVG is read in and compiled (SVG compiler, element 310, after being parsed by the SVG DOM (305), which therefore constitutes interpretation to the binary format (BF) that is output. Next, clearly a scene graph is modified, see for example Steele Figure 7, where this is inherently a scene graph, and data is modified based on these teachings, note as an example the

animation and such, where the position would definitively set – see the entirety of Figure 9, where additionally the visual graph in Figure 9 and the sequence graph exist, where the visual graph is update as items change positions and the like)(Steele clearly shows the user the animated display and the changes in status on display on device 105, see Figure 9 for examples.)

Steele teaches most of the limitations of the instant claim, except for teaching that the function calls are received via an API and that such function calls comprise native format markup language. Steele also does not expressly teach the scene graph, although examiner argues that the objects in Figures 7 and 9 **CLEARLY** constitute a scene graph. Walczak teaches these limitations of the above claim, except for explicitly teaching that VRML uses a scene graph (which is inherent to VRML in any case, and Eleftheriadis teaches that it does 3:1-30) and for suggesting that the scene graph layer utilize an API. The Eleftheriadis reference teaches that the use of APIs for scene graphs is well known in the art and that it is the solution that the prior art had come to with regards to enhancing functionality. Finally, Eleftheriadis itself utilizes plural APIs to access the scene graph, amongst other things (see 4:35-50), where the Scene Graph layer explicitly has its own API (5:37-60). The two references are analogous arts. Next, clearly the X-VRML language is similar to that of the Java language (see Table 1, page 112) in syntax, so the use of similar programmatic structures for the same problem would be extremely obvious, not to mention that X-VRML is implemented in Java (page 117). Therefore, in light of the teachings of Eleftheriadis, it would have been obvious to

one of ordinary skill in the art at the time the invention was made to modify Walczak to utilize an API (possibly that of Eleftheriadis) for the scene graph for at least the reasons already set forth.

Therefore, it would have been obvious to one of ordinary skill in the art to modify the Steele device to use a scripting language variant of SVG (such as X-SVG), where the advantages of such a language are expressly described on page 112 of Walczak, and provide fast and easy access to important functionality without requiring an intermediate language. It further would have been obvious that such a language should use an API to access the scene graph for at least the reasons set forth above.

As to claim 4,

The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking code to place a drawing visual into the scene graph.

Reference Steele teaches this limitation, as for example he teaches the insertion of unique identifiers into media streams [0106], and further [0088] that any visual element or object can modified. Such modifications and insertions *prima facie* must associate code with a suitable / desired insertion as that is the only way either a hierarchy of nodes (e.g. Fig. 7) or single nodes could be logically inserted.

For the second case, if the definition of context is the data associated with a specific element – e.g. the details of the element, its location, color, et cetera, these attributes are inherent in SVG elements as set forth in the rejections above, e.g. sections 11.1, 9.1-9.7, et cetera. Further, Steele teaches the same in Figure 7, where

each element has certain properties that would be a drawing context, in the sense that each visual element has those properties associated with it [Steele 0052-0056 and 0059-0061].

As to claim 5,

The method of claim 4 further comprising, causing drawing context to be returned, the drawing context providing a mechanism for rendering into the drawing visual.

Reference Steele teaches this limitation, as for example he teaches the retrieval of device context in [0101]. Clearly, the device receives information based on its device context, which clearly is associated with the drawing context, as the two are one and the same in this case. Steele teaches rendering in [0007 and 0011-0012]. The drawing context per se is incorporated into the data structures of Steele (see Figure 7). It further would have been obvious to utilize a device specific context so as to optimize data streamed to that device for purposes of minimizing memory consumption (a large problem pointed out by Steele [0007]). (Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task.)

As to claim 6,

The method of claim 2 wherein causing data in the scene graph to be modified comprises invoking code to associate brush data with a visual object in the scene graph.

Reference Steele teaches this limitation via SVG. Turning to the SVG (, section 11 titled 'Painting: Filling, Stroking, and Marker Symbols', specifically section 11.1,

'With SVG, you can paint (e.g. stroke or fill) with: ..." and then proceeds to list several. The term 'brush data' is clearly analogous to the 'paint' operation in SVG with comparable data. Given that SVG allows (from section 11.1) a single color, a solid color (with or without opacity), a gradient, a pattern (vector or image), and custom patterns, clearly each visible element clearly has such data associated with it (see section 11.2 in its entirety, 11.7 for specific properties, section 11.8 for how painting properties can be inherited, which *prima facie* justifies the position that element have intrinsic painting properties, i.e. brush data as set forth above.

As to claim 7,

The method of claim 6 wherein the brush data comprises receiving data corresponding to a solid color.

Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with a solid color with opacity, thus meeting this limitation.

As to claim 8,

The method of claim 6 wherein receiving brush data comprises receiving data corresponding to a linear gradient brush and a stop collection comprising at least one stop.

Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with a gradient that can be linear. Further, sections 11.7.1

and 11.7.2 of the specification sets forth that gradient stops are included in the SVG 'color-interpolation' property.

As to claim 9,

The method of claim 6 wherein receiving brush data comprises receiving data corresponding a radial gradient brush.

Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with a gradient that can be radial and also see sections 11.7.1 and 11.7.2 for more detail, thus meeting this limitation.

As to claim 10,

The method of claim 6 wherein receiving brush data comprises receiving data corresponding to an image.

Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with an image with further details provided in section 11.7.5 under the 'image-rendering' property.

As to claim 11,

The method of claim 10 further comprising, receiving markup corresponding to an image effect to apply to the image.

Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 14.4 of the SVG specification sets

forth that a user can use any image as an opacity mask; thus meeting this limitation, given that alpha blending is *prima facie* an image effect.

As to claim 12,

The method of claim 1 further comprising, receiving markup corresponding to pen data that defines an outline of a shape.

Steele teaches them intrinsically as because of the use of SVG as set forth above in the rejections to claims 4 and 5 and reference SVG clearly supports this position. The term 'pen data' used by applicant above is comparable or analogous to any set of data defining the outline of a shape, including SVG 'path' data. Section 11.3 of the SVG specification sets forth that a user can fill a path that would correspond to the outline of shape with multiple illustrations provided for this under the 'nonzero' and 'even odd' subheadings – see details on paths -- with further details provided in section 11.3 and 11.4 (the individual strokes that create these effects. SVG is a markup language, therefore any SVG rendering utility would *prima facie* receive markup.

As to claim 13,

The method of claim 1 wherein the markup corresponds to a shape class comprising at least one of the set containing rectangle, polyline, polygon, path, line and ellipse shapes.

Reference Steele teaches them intrinsically as set forth above in claim 1 and reference SVG clearly supports this position. The SVG specification sets forth classes of shapes in section 9.1, where all six of the recited shapes (rectangle, polygon, path, line, polyline, and ellipse) are set forth. Further, the SVG view in Steele decomposes

SVG instructions into scene graphs containing basic SVG shapes as above [Steele 0052], where 'Visual Elements' include Shape classes. SVG is a markup language, therefore any SVG rendering utility would *prima facie* receive markup.

As to claim 14,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a geometry-related function to represent a rectangle in the scene graph data structure.

Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. The SVG specification clearly shows in section 9.1 that rectangles are a basic shape, and further that in 9.2 under Example rect02 that such rectangles can have rounded corners, and code is provided that implements such. Also, the 'Rect' class inherently has geometry-related functions as set forth in the beginning to section 9.2. SVG is a markup language, therefore any SVG rendering utility would *prima facie* receive markup. As such, reference Steele shows a rectangle 715 in the scene graph in Figure 7 that intrinsically teaches this limitation. Also, said element can be animated under SVG section 19.2. Steele clearly teaches data modification in [0061] as set forth above.

As to claim 15,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a geometry-related function to represent a path in the scene graph data structure.

Steele Fig. 6 shows an animation where path information is extracted into element 620, and listed in Fig. 7 [see Steele 0050 and 0079]. Further, the SVG specification sets forth path data in section 9.1 as existing and how a 'path' can define a shape or similar. Both meanings are covered herein. Steele clearly teaches data modification in [0061] as set forth above. Itou teaches a path element in 11:15-18.

As to claim 16,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a geometry-related function to represent a line in the scene graph data structure.

Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Further, Steele Fig. 6 shows an animation where path information is extracted into element 620, and listed in Fig. 7 [see Steele 0050 and 0079]. A line element can be animated under SVG section 19.2, which is obviously geometric. Line elements are set forth in SVG section 9.5, and their geometric functions. Steele clearly teaches data modification in [0061] as set forth above.

As to claim 17,

The method of claim 1 wherein the markup is related to hit-testing a visual in the scene graph data structure.

References Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. As such, hit testing would be required for user interactivity, as could the system of Steele under the same rationale. The SVG specification sets forth hit testing in section 16.6 (the two paragraphs right at the end of

the section) where hit testing (namely, hit detection) is taught, specifically testing text for character or cell hits and testing visual elements for hits in their entirety, and such information is clearly communicated in markup language – see section 16.2 for event types and elements transmitted in markup, for example. Steele clearly teaches data modification in [0061] as set forth above.

As to claim 19,

The method of claim 1 wherein the markup is related to calculating a bounding box of a visual in the scene graph data structure.

Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Bounding box calculations are taught in section 7.1 and detailed in section 7.11 where they are calculated. Steele clearly teaches data modification in [0061] as set forth above. In Steele, the scene graph shown in Figure 7 could clearly include lines since they are Visual Elements [Steele 0060-0061, which supports animation, et cetera and would logically include bounding boxes].

As to claim 21,

The method of claim 1 further comprising invoking a visual manager to render a tree of at least one visual object to a rendering target.

Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Further, Steele Fig. 6 shows an animation where path information is extracted into element 620, and listed in Fig. 7 [see Steele 0050 and 0079] as trees. Further, Itou teaches in Fig. 21 the use of trees as set forth there, and teaches manager objects in 2:50-67. SVG teaches that all implementations must

implement a rendering model as set forth in 3.1 and so forth, and scene graphs are known to directed acyclic, i.e. trees. Clearly, this model is implemented through the DOM interfaces set forth in section 4, and each element has its own element information that controls rendering aspects. Steele clearly teaches data modification in [0061] as set forth above. It is *prima facie* obvious that a 'visual manager' of some form must exist in order to handle formatting issues and precedence in rendering, and Steele teaches such a manager in [0075-0076. Clearly the rendering information each visual element [Steele 0056-0061] is sufficient such that it is its own 'rendering target' as set forth above. It would have been obvious to one of ordinary skill in the art to add a manager if necessary to determine the precedence of item rendering with respect to the tree. As such, reference Steele intrinsically teaches this limitation.

As to claim 22,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place a container object in the scene graph data structure, the contained object configured to contain at least one visual object.

Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Further, Steele Fig. 7 shows a tree derived from an animation is shown – Figure 6 [see Steele 0050 and 0079]. SVG clearly teaches the use of container objects, as in section 1.6 it clearly teaches the use of 'container elements', which are defined as 'An element that can have graphic elements and other container elements as child elements'. Steele clearly teaches data modification in [0061] as set forth above. Clearly, the container object could be the

head object of the tree structure shown in Steele Fig. 7 or the group node of Itou, shown as the head node in Fig. 21. As such, reference Steele intrinsically teaches this limitation.

As to claim 23,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place image data into the scene graph data structure.

Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Clearly, visual elements can be covered by or tiled with images as established in SVG section 11.1, where SVG teaches: "...can paint (i.e. fill or stroke) with: ...a pattern (vector or image, possibly tiled) ..." which clearly establishes this, with more detail in section 11.7.5 and 11.12.

As to claim 24,

The method of claim 23 wherein causing data in the scene graph to be modified comprises invoking a function to place an image effect object into the scene graph data structure that is associated with the image data.

Reference Steele teaches SVG. Section 14.4 of the SVG specification sets forth that a user can use any image as an opacity mask for any visual element, thus meeting this limitation, given that alpha blending is *prima facie* an image effect. Steele clearly teaches data modification in [0061] as set forth above. As such, references Itou and Steele intrinsically teach this limitation. Motivation for combining Itou, Kim, and SVG is taken from the rejection to claim 1. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 25,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place data corresponding to text into the scene graph data structure.

Reference Steele teaches SVG and clearly supports this position. Section 10.1 of the SVG specification sets forth the use of a 'text' element, and Steele teaches the inclusion of text element 725 in the data tree shown in Fig. 7. Steele clearly teaches data modification in [0061] as set forth above.

As to claim 26,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to provide a drawing context in response to the function call.

Reference Steele teaches this limitation, as for example he teaches the retrieval of device context in [0101]. Clearly, the device receives information based on its device context, which clearly is associated with the drawing context, as the two are one and the same in this case. For the second case, if the definition of context is the data associated with a specific element – e.g. the details of the element, its location, color, et cetera, these attributes are inherent in SVG elements as set forth in the rejections above, e.g. sections 11.1, 9.1-9.7, et cetera. Further, Steele teaches the same in Figure 7, where each element has certain properties that would be a drawing context, in the sense that each visual element has those properties associated with it [Steele 0052-

0056 and 0059-0061]. Steele clearly teaches data modification in [0061] as set forth above.

As to claim 27,

The method of claim 26 wherein the function call corresponds to a retained visual, and further comprising, calling back to have the drawing context of the retained visual returned to the scene graph data structure.

Steele teaches SVG. Section 11.1 of the SVG specification clearly sets forth that all elements (as well as 3.1 and 4.2) have properties associated with them. The system of Steele clearly caches visuals during processing – see [0083], and it would be obvious that such data would be pulled from the cache to find out the state and properties of a visual element. Steele clearly teaches data modification in [0061] as set forth above. Further, it would be obvious to one of ordinary skill to so modify Steele reference to cache the visuals so that they would be retained and that data would be pulled from the cache as set forth above, and as the Steele reference sets forth to have it pulled from there during data processing, including that of data trees like unto the one in Figure 7, as in [0100 Steele]. As such, references Itou and Steele intrinsically teach this limitation. Motivation for combining Itou, Kim, and SVG is taken from the rejection to claim 1. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 30,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place animation data into the scene graph data structure.

Reference Steele does teach it. Steele in Figs. 6 and 9 shows animated elements [0041] and in Fig. 7 shows that each subgroup is shifted a certain amount with x and y coordinates given. Steele [0050, 0052] for example provides that such animation takes place, and the SVG standard in 19.1 – 19.5 clearly sets forth how each element can have animation associated with it, which clearly is placed into the scene graph of Fig. 7. Therefore, clearly animation data is put into the tree of Fig. 7 Steele, which is clearly a scene graph by every known definition of the term, and a sample SVG XML program is provided in the second page of Fig. 9.

As to claim 31,

The method of claim 30 further comprising communicating timeline information corresponding to the animation data to a composition engine.

Reference Steele does teach the animation limitation. Steele clearly establishes in [0051-0054] and Figs. 6 and 9 that animation takes place through the SVG standard. Section 19.2 of SVG sets forth how this takes place, and at the bottom three paragraphs of section 19.2.2 it clearly states that animation has a document start and document end, and further in the second to last paragraph that the SVG system indicates the timeline position of document fragments being animated. Further, according to SVG 19.2.2 the animation is by document fragment and object path, which clearly are passed to the system is specified in, for example, the second page of Fig. 9 in the SVG XML program. Clearly, the system of Steele performs compositing and rendering [0007, 0011-0012]. Finally, reference SVG teaches that it supports compositing (section 14.2.1), and the X3D specification supports compositing (6.2.3 for example). The

composition engine would be, for example, elements 143, 144, and 147 of Fig. 2 of Kim. Motivation for combining Itou, Kim, and SVG is taken from the rejection to claim 1. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 32,

The method of claim 31 wherein the composition engine interpolates graphics data based on the timeline to animate an output corresponding to an object in the scene graph data structure.

Reference Steele does teach it. Steele in Figs. 6 and 9 shows animated elements [0041] and in Fig. 8, clearly during an animation the actions are shown, where the system in steps 835, 840, and 845 performs interpolation for nodes shown in the tree in Fig. 7. Clearly interpolation takes place during animation [0072, 0077-0079] which performs interpolation during the animation process as set forth in the SVG standard, and *prima facie* the output would only be objects in the scene graph, and they would *prima facie* be based on the timeline as set forth in the rejection to claim 31 above.

As to claim 33,

The method of claim 32 wherein the composition engine is at a low-level with respect to the scene graph.

References Itou and Kim do not expressly teach this limitation, while reference Steele does teach it. Steele in Fig. 15 shows that various programs, including the operating system, are on the flash memory, which in [0136] is specified to contain all the low-level programs of the operating system – graphics is low-level functionality. Since

there is no specific graphics unit, all graphics operations and compositing are done by the operating system in the microprocessor, which *prima facie* means that in that embodiment, such graphics are done at a low-level, that is the rendering is done by the operating system at a low level. The scene graph is high-level in that it is embodied in RAM and is held as an abstraction – this is a function of the SVG standard that keeps tree nodes and container nodes as abstract as possible, therefore the embodiment in Fig. 18 must do the same. In any case, low-level composition means that it would be done by hardware that is much faster than software. As such, it would be obvious to modify the device of Kim and Steele to use low-level rendering, and in any case Steele has the rendering means set forth in the rejection to claim 32 above, which is *prima facie* entirely hardware.

As to claim 34,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place an object corresponding to audio and/or video data into the scene graph data structure.

Reference Steele does teach it. Steele in Figs. 8 shows audio elements 820 and 830 in the animation execution and in Fig. 7 a scene graph data structure (a tree). Steele [0050, 0052] for example provides that such animation takes place, and the SVG standard in 6.18 clearly sets forth aural style sheets, that are audio data that each element can have animation associated with it, which clearly is placed into the scene graph of Fig. 7. Also, by definition, SVG animations would be video.

As to claim 35,

The method of claim 1 wherein causing data in the scene graph to be modified comprises changing a mutable value of an object in the scene graph data structure.

Steele teaches in [0014] that one embodiment of his invention changes the visual graph in accordance to changes of the sequence graph, where the visual graph is comparable to the “scene graph” of applicant and mutable values (e.g. position) of elements in the tree are shifted as per Steele [0052-0057]. Therefore, the limitation is met, and it would have been obvious to modify it so that it in fact change mutable values on the tree if applicant feels that this is not an adequate embodiment of this particular limitation.

Conclusion

Applicant is asked to carefully consider the Steele reference in light of the Walczak paper for its teaching that markup languages should advantageously be in script format and be object-oriented, as well as the Eleftheriadis publication stating that APIs are advantageous. Examiner respectfully submits that this particular combination

teaches the core concepts of applicant's invention, and applicant is advised to carefully tailor the next amendment to avoid it.

Applicant is also asked to carefully consider this reference:

US 2004/0015740 and US 2004/0039496 to Dautelle et al, which both discuss a two-dimensional API with a scene graph, which would seem to be a good combination with the Steele reference for another round of rejections of all claims under 35 USC 103(a). Applicant is advised that such a separate set of rejections against all claims will probably be placed in the next Office Action, so applicant is kindly requested to provide a section in the reply addressing why such a combination would not be valid against the claims, thus obviating the need for examiner to add such a separate set of claims.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Eric Woods whose telephone number is 571-272-7775. The examiner can normally be reached on M-F 7:30-5:00.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Ulka Chauhan can be reached on 571-272-7782. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

Eric Woods

August 2, 2006

Ulka Chauhan

ULKA CHAUHAN
SUPERVISORY PATENT EXAMINER